Performance Working Group
Grid Working Document: GWD-Perf-13-1
Category: Informational
Obsoletes: GWD-Perf-2-1

Warren Smith
NASA Ames Research Center
Dan Gunter
Lawrence Berkeley National Laboratory
June 6 2001

Simple LDAP Schemas for Grid Monitoring

1 Introduction

The purpose of this document is to provide an initial definition of the data we need in a directory service or database so that we can implement a performance monitoring testbed. To begin with, this document describes how to represent producers of events and event schemas. The representation of producers is simple and does not contain information such as who has access to the events and what protocols can be used to access the events. In the future, we will define how to describe consumers of events and add details to our representations.

A popular choice for a directory service or database for grid computing is a distributed directory service that is accessed using the Lightweight Directory Access Protocol (LDAP) [2, 3]. This document uses LDAP terminology, schemas, and formats to represent the directory service schemas.

The next section provides some background information. Section 3 describes how providers of events can advertise their existence and the types of events they provide, and how consumers can find providers using the LDAP protocol. Section 3.2 discusses how event schemas can be stored in a LDAP directory.

2 Background

We define a very basic architecture in the Grid Monitoring Service Architecture working document [4]. This architecture is shown in Figure 1 and consists of three components. An *event producer* is an entity such as a host monitor or application performance monitor that is generating events. An *event consumer* is an entity that wishes to receive events from a producer. Finally, the *directory service* is used for consumers to find producers. The figure also mentions the most important piece of data in our design: *events*. Events are time-stamped sets of information that are generated by producers and sent to consumers. The figure also illustrates what interfaces and data we need to define. We need to define the interface between producers and consumers. This interface consists of the protocol they use to communicate. The interface to the directory service is LDAP but we must define how our data is stored in the directory service. That is the focus of this document.

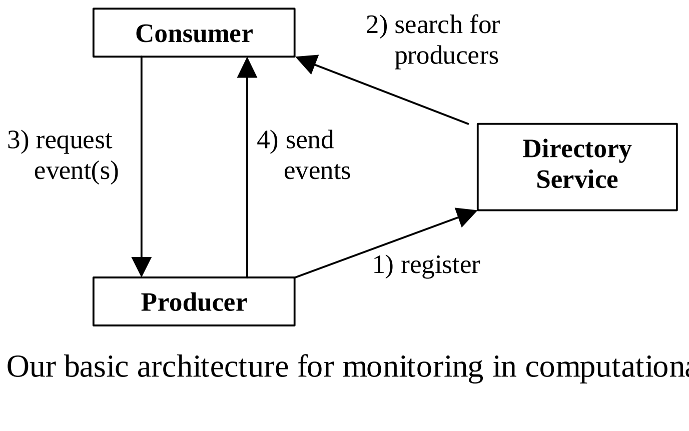

Figure 1. Our basic architecture for monitoring in computational grids.

At this point, we do not plan to store events in the information service. However, events are one of the central components of our monitoring architecture so it is good to define exactly what we think they are. An event consists of:

1. A name that identifies the type of the event.
2. A set of elements. Each element has a unique name within the event and one or more typed values. Our current idea is to only have simple types. That is, types such as char, short, long, float, double, string, and so on. The consensus seems to be that having more complex types is not necessary at the current time. Each value may also have units and accuracy associated with it. There are several elements that are part of every event such as a timestamp.

Each event has a name. The name identifies an *event schema*. An event schema constrains an event's elements and the types of those elements. An event schema consists of a name and one or more element schemas. Each element schema consists of:

1. A name,
2. a type,
3. a text description,
4. the minimum number of occurrences of the element in an event of this type, and
5. the maximum number of occurrences of the element in an event of this type.

To allow events to be understood, event schemas and their associated information must be propagated to consumers so that consumers can determine what events contain the information they need. One way we wish to support this propagation is to provide an *event dictionary*. An event dictionary is simply a listing of the common event schemas that are defined by various organizations. Such event dictionaries can be stored in an LDAP-based information service and we will address the issue of how this can be accomplished in this document. LDAP may be a useful way to store event dictionaries because an LDAP server can provide a structured set of event type information that is easily searched or dumped using LDAP client libraries.

We also define *event parameters* that are the information a consumer provides to a producer to describe the events it is interested in. For example, a consumer may want to know information about the status of a process. The consumer must therefore provide information such as the ID of the process to monitor and how often to monitor it. The parameters are passed to a producer and the producer uses this information to determine how and which events to generate. Each parameter description consists of

1. A name,
2. a type,
3. a text description,

6. the minimum number of occurrences of the element in an event of this type, and
7. the maximum number of occurrences of the element in an event of this type.

2.1 LDAP

Projects to provide computational grid services and recent activity in the Grid Forum indicate that the community is adopting distributed databases accessed using the Lightweight Directory Access Protocol as a primary source of information in computational grids. LDAP is a standard protocol to access directory services that has become a de facto standard. It was originally intended to access X.500 directories using a simpler interface than the X.500 Directory Access Protocol (DAP). While very few of the current LDAP servers are implemented using X.500, this original intention of LDAP has defined some of the assumptions about how the data accessed through LDAP is organized.

LDAP assumes that the entries in the database are organized in a tree, similar to the DNS namespace. A LDAP namespace has a root and for computational grids, the root we use is “o=Grid”. Our namespace, called the Directory Information Tree (DIT) branches out below this root by adding organizations, organizational units, resources, etc. An example DIT is shown in Figure 2. The unique name for the “service=JobManager” entry on the lower left side is “service=JobManager, hn=evelyn.nas.nasa.gov, ou=Ames Research Center, o=National Aeronautics and Space Administration, o=Grid”. It’s a rather long unique name, but you can see that it is formed by walking up the tree to the root. To use a LDAP directory service, we must define where our monitoring entries should be placed in this DIT.

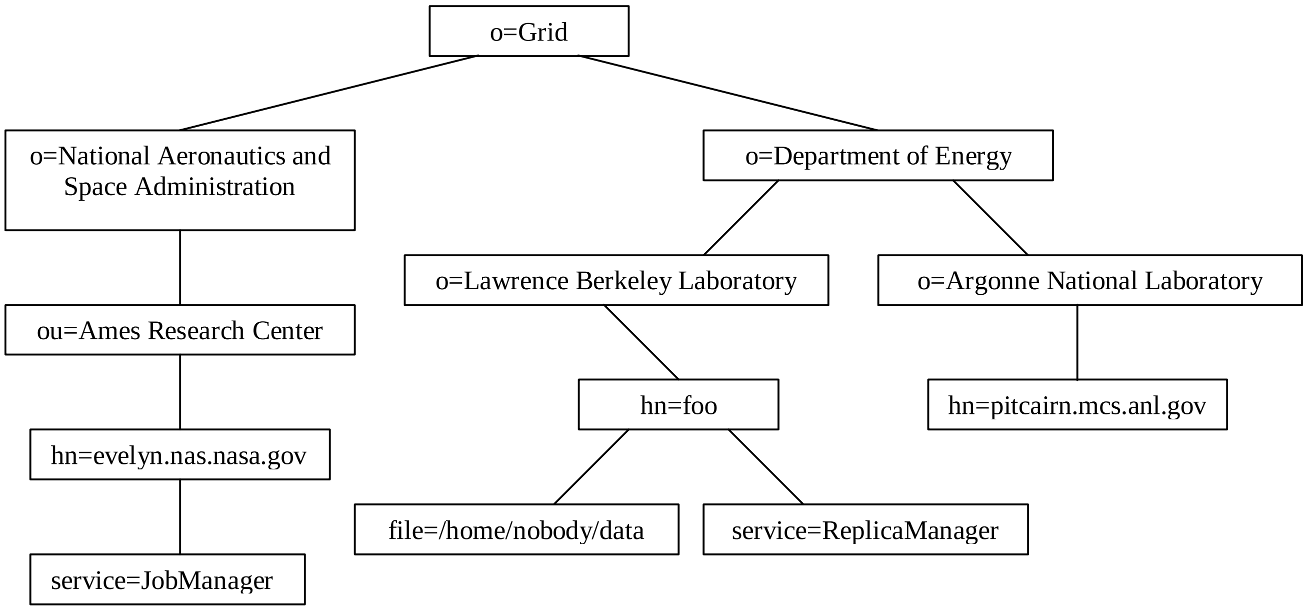

Figure 2. An example directory information tree.

All of the entries in an LDAP database are defined to be instances of one or more “types.” The LDAP “types” are called object classes. An object class consists of a name, all of the attributes (<name, value> pairs) that must be present in an instance of that object class, and all of the attributes that may be present in an instance of that object class. An object class can also be derived from an existing object class. This means that the derived object class contains all of the must and may elements of its parent. Other must and may elements can then be added in the child class. Further, an entry in the database can be of several different object classes. This is useful, for example, if you have entries for computer systems. They may all be of the object class ComputeResource, but some may also have benchmark information. These entries can then be of object class ComputeResource and BenchmarkInformation.

3 Representation of Event Producers

This section describes the LDAP object classes and the directory information tree we propose to represent producers of events for our performance monitoring testbed.

3.1 LDAP Object Classes

For our initial representations of producers, we use three object classes:

EventProducer. The EventProducer class is the object class associated with an event Producer. It contains information about how to contact the producer.

EventInstance. The EventInstance class is used to describe the events available from a producer. For example, an instance might describe which UptimeCPULoad events are available from a producer.

ElementInstance. The ElementInstance class is used to describe an element that will be part of an event or the parameters specified when requesting an event. For example, an element instance along with an event instance might say that this producer can provide UptimeCPULoad events that have an output element of HostName of foo.nas.nasa.gov and an input Period of at least 15 seconds. In the future, a unified way of specifying wildcards or ranges may be proposed, but for now this can be decided one event/element at a time and documented in the appropriate EventSchema/EventElementSchema.

Table 1 shows the information in an EventProducer object class. We have combined a protocol name and version in the "Protocol" attribute because a producer may support more than one protocol. If the name and version of each protocol are in different attributes, it may be difficult to match a name with a version if there is more than 1 protocol supported. Note that these definitions are relatively simple and will need to be extended in the future. For example, no information is provided about which users have access to what events.

Table 1. eventProducer LDAP object class.

Name	**Type**	**Required or Optional**	**Number of Values**	**Description**
producerName	String	Required	1	A unique name for this event producer under the entity where it sits in the DIT.
hostName	String	Required	1	The host name the Producer is running on.
port	Integer	Required	1	The port the Producer is listening to.
protocol	String	Required	1+	A protocol that the producer supports. This value includes a name and a version separated by a space. The only value that is currently valid is "XML b1.0".
description	String	Optional	1	A textual description of the event producer.

Table 2. eventInstance LDAP object class.

Name	**Type**	**Required or Optional**	**Number of Values**	**Description**
instanceName	String	Required	1	The name of this event instance. There may be multiple instances of the same event available from a producer so the eventName cannot be used as the unique name in the DIT.
eventName	String	Required	1	The name of the event.
inputNameSpace	String	Required	1	The name space used for the "input": the parameters specified when requesting the event.
outputNameSpace	String	Required	1	The name space used for the "output": the event and the elements of the event.
eventLocation	Reference	Required	1	The location of the schema for the event in the LDAP DIT.

Table 3. elementInstance LDAP object class.

Name	Type	Required or Optional	Number of Values	Description
elementName	String	Required	1	The name of the element.
relation	String	Optional	1	The relation of the value in the event or event parameters to the value provided in the value attribute. The possible relations are =, !=, <, <=, >, and >=.
value	String	Optional	1	The value of this element in every event that is generated that matches this instance.
units	String	Optional	1	The units for the value.
accuracy	String	Optional	1	The accuracy of the value.
inputOutput	String	Required	1	If this element is for input, output, or input/output.

3.2 LDAP Directory Information Tree

The sub-trees for producer information in the DIT are relatively small. Each EventProducer has 1 or more EventInstance children and each EventInstance has one or more ElementInstance children. An example of this subtree is shown in Figure 3. This figure shows a producer that can provide UptimeCPULoad events about the host foo.nas.nasa.gov and Ping events from the host foo.nas.nasa.gov. Section 6 will discuss the overall DIT for all of our objects.

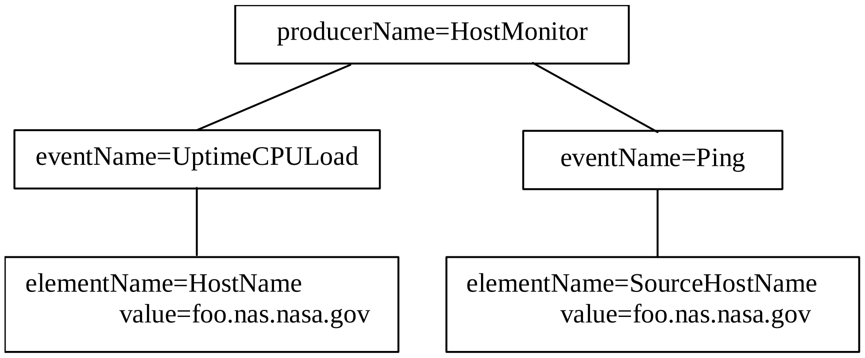

Figure 3. Directory information sub-tree for event producers.

4 Representation of Event Consumers

We also wish to represent event consumers in LDAP so that a consumer can advertise itself and producers can find it. The difficulty is that we have not thought very much about what the typical consumers will be and it is therefore difficult to represent them in a directory service. For example, consumers may accept only certain kinds of events, may only be available to certain users, may archive events or only store the most recently received event from each subscription, and may allow the events to be accessed in a stream or searched in various ways. We avoid this problem for now by simply registering the existence of a consumer and providing no information about the capabilities of the consumer.

4.1 LDAP Object Classes

Table 4. eventConsumer LDAP object class.

Name	Type	Required or Optional	Number of Values	Description
consumerName	String	Required	1	A unique name for this event consumer under the entity where it sits in the DIT.
hostName	String	Required	1	The host name the consumer is running on.
port	Integer	Required	1	The port the consumer is listening to.
protocol	String	Required	1+	A protocol that the consumer supports in the form of a URI. This URI is of the form http://<defining organization info>/<protocol name>/<protocol version>. The only protocol we have defined so far is http://www.gridforum.org/ Performance/ ProducerConsumerProtocol/ XML/b1.0.
description	String	Optional	1	A textual description of the event consumer.

4.2 LDAP Directory Information Tree

The sub tree for each event consumer is trivial and only consists of an EventConsumer object. The locations of these objects in the overall directory information tree is described in Section 6.

5 Representation of Event Schemas

This section describes the LDAP object classes and the directory information tree we propose to represent event schemas for our performance monitoring testbed.

5.1 LDAP Object Classes

There are two obvious ways that we could represent event schemas using LDAP. The first technique is to have a single entry for each event schema. That entry would consist of, say, two attributes. The first attribute would contain the name of the schema. The second attribute would contain the schema represented in, say, XML Schema [1]. The advantage of this approach is that it keeps the LDAP information simple. The disadvantage is that all producers and consumers of events must understand XML Schema. While the first event protocol we are working on uses XML and XML Schema, we believe that we will define other protocols that will not use XML. It does not seem to make much sense to force implementers of these later protocols to include XML tools in their implementations just to read and write event schemas.

We choose the second obvious approach that is to represent event schemas directly using LDAP entries. If our schemas can be represented this way, all we require is that producers and consumers be able to understand the LDAP protocol (which we already require) and that they understand how we represent event schemas in LDAP.

We currently plan to use two LDAP object classes to represent schemas for events and event parameters. They are:

EventSchema. The EventSchema class provides a description of one type of event that is provided by any producer.

ElementSchema. The ElementSchema object class describes one element that may be contained in an event of this schema or in the parameters specified when requesting events. Instances of this class are associated with EventSchema instances.

Table 5. eventSchema LDAP object class.

Name	**Type**	**Required or Optional**	**Number of Values**	**Description**
eventName	String	Required	1	The name of the event.
inputNameSpace	String	Required	1	The name space used for the parameters specified when requesting an event, the input.
outputNameSpace	String	Required	1	The name space used for the event, the output.
description	String	Optional	1	A textual description of the event schema.

Table 6. ElementSchema LDAP object class.

Name	**Type**	**Required or Optional**	**Number of Values**	**Description**
elementName	String	Required	1	The name of the element.
nameSpace	String	Required	1	The name space of the element.
type	String	Required	1	The type of the element (char, short, int, long, float, double, unsignedChar, unsignedShort, unsignedInt, unsignedLong, longDouble, string). What types do we want to use?!?
units	String	Optional	1	The default units for the element.
accuracy	String	Optional	1	The default accuracy for the element.
minOccurances	Integer	Optional	1	The minimum number of times this element must appear in the event it is associated with. No value means that the minimum number of occurrences is 0.
maxOccurances	Integer	Optional	1	The maximum number of times this element can appear in the event it is associated with. No value or a negative number indicates an unbounded maximum.
inputOutput	String	Required	1	If the element is used in the parameters specified when requesting an event (input), the event itself (output), or both. The valid values are input, output, or input/output.
description	String	Optional	1	A textual description of the element schema.

5.2 LDAP Directory Information Tree

Figure 4 shows the obvious arrangement of EventSchema and ElementSchema objects in the directory information tree. As you can see, each EventSchema object has one or more ElementSchema objects as children and these ElementSchema objects describe an event and the parameters that can be specified when asking for the event.

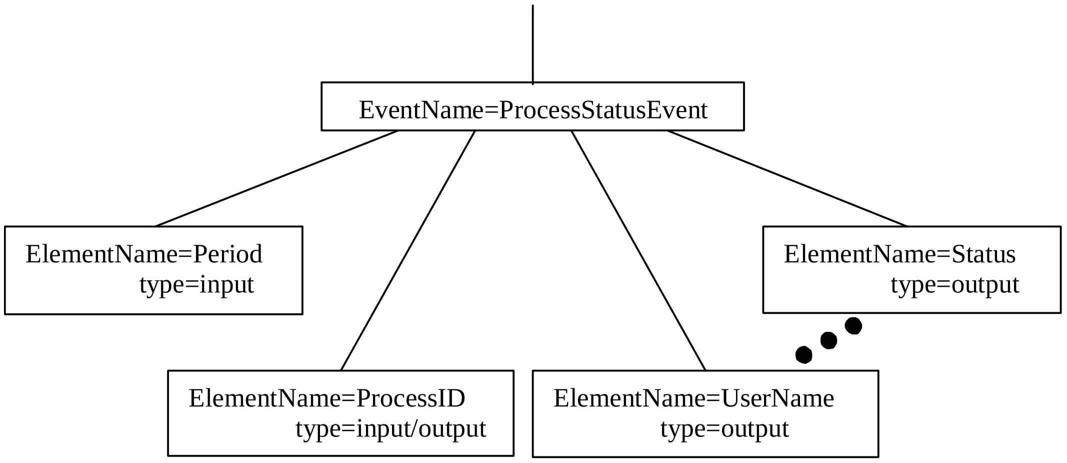
EventName=ProcessStatusEvent
ElementName=Period type=input
ElementName=Status type=output
ElementName=ProcessID type=input/output
ElementName=UserName type=output

Figure 4. Event type sub-tree in the directory information tree.

6 LDAP Directory Information Tree

While Sections 3.2 and 5.2 described the DIT structure of the producer and schema sub-trees, this section describes the overall structure of the directory information tree that we are using. The question we must address is where should the sub-trees rooted at EventProducer and EventSchema instances be placed? The EventSchema instances are not associated with any particular host or producer, but they are associated with an organization that defines them. This leads us to propose that EventSchema sub-trees be placed in the DIT under the organizations that define them. EventProducer instances are running on a host so they are associated with both a host and an organization. The EventProducer sub-trees can therefore be children of either host instances or organizational instances. Since we don't have a need to define host object classes for our testbed, we propose that EventProducer sub-trees be children of organizational instances.

The final question to answer is what is the root and high-level structure of our information in the DIT? The selection of a root entry can be a complex issue (of which we do not know all of the ramifications), but since the grid community seems to be rooting their information at "o=Grid", we will do the same. We propose, similar to the way things are done now, that the high-level structure of our DIT follow our real-life organizational structure. Some examples:

- ou=Performance Working Group, o=Grid Forum, o=Grid
- ou=Ames Research Center, o=National Aeronautics and Space Administration, o=Grid
- o=Lawrence Berkeley National Laboratory, o=Department of Energy, o=Grid

The "o" above stands for organization, and the "ou" stands for organizational unit. These are standard attributes with associated object classes that are defined by LDAP. An example of a DIT is shown in Figure 5.

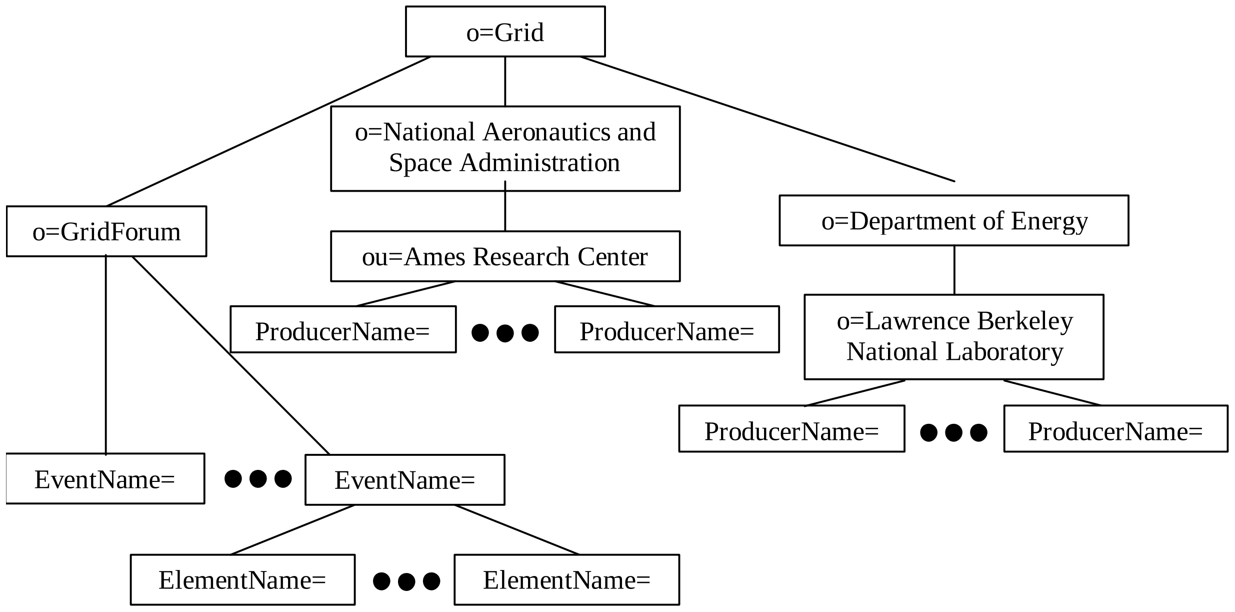

Figure 5. Directory Information Tree including entries for monitoring information.

References

[1] D. Fallside, "XML Schema Part 0: Primer," http://www.w3.org/TR/xmlschema-0/.

[2] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*: Macmillan Technical Publishing, 1997.

[3] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*: MacMillan Technical Publishing, 1999.

[4] B. Tierney, R. Aydt, D. Gunter, W. Smith, Valerie Taylor, R. Wolski, and M. Swany, "A Grid Monitoring Service Architecture," Global Grid Forum Performance Working Group 2001.